Bộ môn Công nghệ Phần mềm
Viện CNTT & TT
Trường Đại học Bách Khoa Hà Nội

LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG
Bài 07. Đa hình (Polymophism)

Nội dung

1. Upcasting và Downcasting
2. Liên kết tĩnh và Liên kết động
3. Đa hình (Polymophism)
4. Lập trình tổng quát (generic prog.)

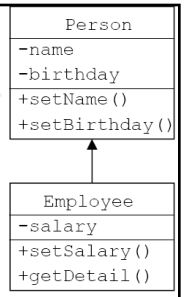
Nội dung

1. **Upcasting và Downcasting**
2. Liên kết tĩnh và Liên kết động
3. Đa hình (Polymophism)
4. Lập trình tổng quát (generic prog.)

1.1. Upcasting

- Moving up the inheritance hierarchy
- Up casting là khả năng nhìn nhận đối tượng thuộc lớp dẫn xuất như là một đối tượng thuộc lớp cơ sở.
- Tự động chuyển đổi kiểu

Ví dụ




```
public class Test1 {
  public static void main(String arg[]){
      Person p;
      Employee e = new Employee();
      p = e;
      p.setName("Hoa");
      p.setSalary(350000); // compile error
}
```

5

Ví dụ (2)

```
class Manager extends Employee {
  Employee assistant;
  // ...
  public void setAssistant(Employee e) {
      assistant = e;
  }
  // ...
}
public class Test2 {
  public static void main(String arg[]){
      Manager junior, senior;
      // ...
      senior.setAssistant(junior);
  }
}
```

6

Ví dụ (3)

```
public class Test3 {
  String static teamInfo(Person p1, Person p2){
      return "Leader: " + p1.getName() +
             ", member: " + p2.getName();
  }

  public static void main(String arg[]){
      Employee e1, e2;
      Manager m1, m2;
      // ...
      System.out.println(teamInfo(e1, e2));
      System.out.println(teamInfo(m1, m2));
      System.out.println(teamInfo(m1, e2));
  }
}
```

7

1.2. Downcasting

- Move back down the inheritance hierarchy
- Down casting là khả năng nhìn nhận một đối tượng thuộc lớp cơ sở như một đối tượng thuộc lớp dẫn xuất.
- Không tự động chuyển đổi kiểu
  → Phải ép kiểu.

8

Ví dụ

```
public class Test2 {
  public static void main(String arg[]){
      Employee e = new Employee();
      Person p = e; // up casting
      Employee ee = (Employee) p; // down casting
      Manager m = (Manager) ee; // run-time error

      Person p2 = new Manager();
      Employee e2 = (Employee) p2;

      Person p3 = new Employee();
      Manager e3 = (Manager) p3;
  }
}
```

9

Nội dung

1. Upcasting và Downcasting
2. **Liên kết tĩnh và Liên kết động**
3. Đa hình (Polymophism)
4. Lập trình tổng quát (generic prog.)

10

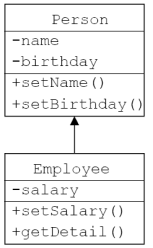
2.1. Liên kết tĩnh (Static Binding)

- Liên kết tại thời điểm biên dịch
  - Early Binding/Compile-time Binding
  - Lời gọi phương thức được quyết định khi biên dịch, do đó chỉ có một phiên bản của phương thức được thực hiện
  - Nếu có lỗi thì sẽ có lỗi biên dịch
  - Ưu điểm về tốc độ

11

Ví dụ

```
public class Test {
  public static void main(String arg[]){
      Person p = new Person();
      p.setName("Hoa");
      p.setSalary(350000); //compile-time error
  }
}
```



12

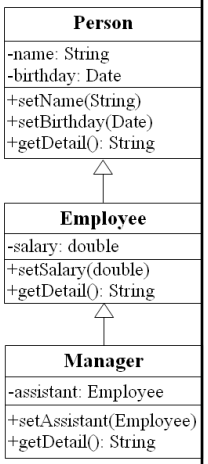
2.2. Liên kết động (Dynamic binding)

- Lời gọi phương thức được quyết định khi thực hiện (run-time)
  - Late binding/Run-time binding
  - Phiên bản của phương thức phù hợp với đối tượng được gọi.
  - Java mặc định sử dụng liên kết động

13

Ví dụ

```
public class Test {
  public static void main(String arg[]){
    Person p = new Person();
    // ...
    Employee e = new Employee();
    // ...
    Manager m = new Manager();
    // ...
    Person pArr[] = {p, e, m};
    for (int i=0; i< pArr.length; i++){
      System.out.println(
            pArr[i].getDetail());
    }
  }
}
```



14

Nội dung

1. Upcasting và Downcasting
2. Liên kết tĩnh và Liên kết động
3. **Đa hình (Polymophism)**
4. Lập trình tổng quát (generic prog.)

15

3. Đa hình (Polymophism)

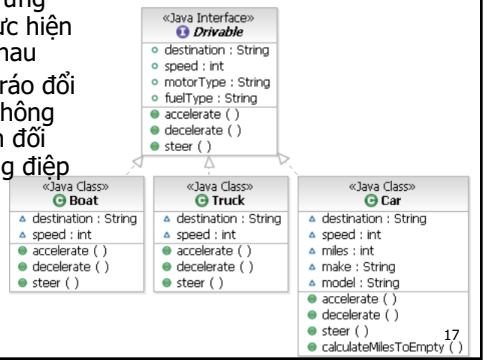
- Ví dụ: Nếu đi du lịch, bạn có thể chọn ô tô, thuyền, hoặc máy bay
  - Dù đi bằng phương tiện gì, kết quả cũng giống nhau là bạn đến được nói cần đến
  - Cách thức đáp ứng các dịch vụ có thể khác nhau



16

3. Đa hình (2)

- Các lớp khác nhau có thể đáp ứng danh sách các thông điệp giống nhau, vì vậy cung cấp các dịch vụ giống nhau
  - Cách thức đáp ứng thông điệp, thực hiện dịch vụ khác nhau
  - Chúng có thể tráo đổi cho nhau mà không ảnh hưởng đến đối tượng gửi thông điệp
  - → Đa hình



17

3. Đa hình (3)

- Polymorphism: Nhiều hình thức thực hiện, nhiều kiểu tồn tại
- Đa hình trong lập trình
  - Đa hình phương thức:
    - Phương thức trùng tên, phân biệt bởi danh sách tham số.
  - Đa hình đối tượng
    - Nhìn nhận đối tượng theo nhiều kiểu khác nhau
    - Các đối tượng khác nhau cùng đáp ứng chung danh sách các thông điệp có giải nghĩa thông điệp theo cách thức khác nhau.

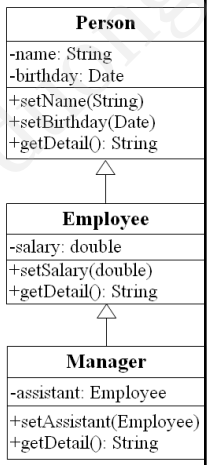
18

3. Đa hình (4)

- Nhìn nhận đối tượng theo nhiều kiểu khác nhau → Upcasting và Downcasting

```
public class Test3 {
  public static void main(String
                      args[]){
    Person p1 = new Employee();
    Person p2 = new Manager();

    Employee e = (Employee) p1;
    Manager m = (Manager) p2;
  }
}
```



19

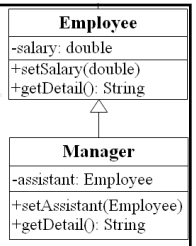
3. Đa hình (5)

- Các đối tượng khác nhau giải nghĩa các thông điệp theo các cách thức khác nhau → Liên kết động
- Ví dụ:

```
Person p1 = new Person();
Person p2 = new Employee();
Person p3 = new Manager();
// ...
System.out.println(p1.getDetail());
System.out.println(p2.getDetail());
System.out.println(p3.getDetail());
```

20

Ví dụ khác




```
class EmployeeList {
  Employee list[];
  ...
  public void add(Employee e) {...}
  public void print() {
    for (int i=0; i<list.length; i++) {

    System.out.println(list[i].getDetail());
    }
}
 ...
  EmployeeList list = new EmployeeList();
  Employee e1; Manager m1;
  ...
  list.add(e1); list.add(m1);
  list.print();
```

21

Toán tử instanceof

```
public class Employee extends Person {}
public class Student extends Person {}

public class Test{
  public doSomething(Person e) {
    if (e instanceof Employee) {...
    } else if (e instanceof Student) {... ){
    } else {...}
  }
}
```

22

Static Binding và Polymorphism

- Function/Method Overloading
- Constructor Overloading: là 1 thể hiện của function overloading
- Operator Overloading

- C# và C++ cư xử với hàm định nghĩa chồng theo cơ chế static binding trong khi Java cư xử theo cơ chế dynamic binding

23

Dynamic Binding và Polymorphism

- Abstract Class
- Overriding:
  - Java: mặc định phương thức lớp con trùng chữ ký lớp cha → là overriding
  - C#: Muốn overriding một phương thức ảo hoặc trừu tượng thuộc lớp cha phải dùng từ khoá override
  - C++: phương thức của lớp dẫn xuất cùng kí hiệu với phương thức được khai báo với từ khoá virtual trong lớp cha → nó overriding phương thức của lớp cha (tương tự Java)
- Abstract Method và Virtual Method
  - Abstract method: là p/thức chỉ có khai báo interface, không có cài đặt. Lớp dẫn xuất từ lớp có các abstract method phải implement tất cả các phương thức loại này. Java, C#: dùng từ khóa abstract
  - Virtual method: Là p/thức có thể bị overridden trong lớp dẫn xuất và phải có cài đặt trong lớp cơ sở. Lớp dẫn xuất có thể không cần overriding p/thức này.
    - Java: mặc định mọi phương thức đều là virtual (trừ final)
    - C#: khai báo dùng từ khóa virtual. Dùng từ khóa override cho phương thức ghi đè ở lớp dẫn xuất.
      - Riêng với C#, nếu bỏ 2 từ khóa → che được phương thức lớp cha. Để tránh warning, dùng từ khóa new chỉ tường minh (chuyển thành static binding)
    - C++: khai báo dùng từ khóa virtual. Ý nghĩa như Java, mặc định bị ghi đè, không che được

24

Nội dung

1. Upcasting và Downcasting
2. Liên kết tĩnh và Liên kết động
3. Đa hình (Polymophism)
4. **Lập trình tổng quát (generic prog.)**

25

4. Lập trình tổng quát

- 4.1. Giới thiệu
- 4.2. Java generic data structure
  - 4.2.1. Data structure
  - 4.2.2. Java collection framework
  - 4.2.3. Các interface trong Java collection framework
  - 4.2.4. Các cài đặt cho các interface – implementation
- 4.3. Định nghĩa và sử dụng Template
- 4.4. Ký tự đại diện (Wildcard)

26

4. Lập trình tổng quát

- **4.1. Giới thiệu**
- 4.2. Java generic data structure
  - 4.2.1. Data structure
  - 4.2.2. Java collection framework
  - 4.2.3. Các interface trong Java collection framework
  - 4.2.4. Các cài đặt cho các interface – implementation
- 4.3. Định nghĩa và sử dụng Template
- 4.4. Ký tự đại diện (Wildcard)

27

4. 1. Giới thiệu về lập trình tổng quát

- Tổng quát hóa chương trình để có thể hoạt động với các kiểu dữ liệu khác nhau, kể cả kiểu dữ liệu trong tương lai
  - thuật toán đã xác định
- Ví dụ:
  - C: dùng con trỏ void
  - C++: dùng template
  - Java: lợi dụng upcasting
  - Java 1.5: template

28

Ví dụ: C dùng con trỏ void

- Hàm memcpy:

```
void* memcpy(void* region1,
             const void* region2, size_t n){
  const char* first = (const char*)region2;
  const char* last = ((const char*)region2) + n;
  char* result = (char*)region1;
  while (first != last)
     *result++ = *first++;
  return result;
}
```

29

Ví dụ: C++ dùng template

```
template<class ItemType>
void sort(ItemType A[], in
  // Sort count items in the arr
  // The algorithm that is us
  for (int i = count-1; i > 0; i--) {
     int index_of_max = 0;
     for (int j = 1; j <= i ; j++)
        if (A[j] > A[index_of_max]) index_of_max = j;
     if (index_of_max !=  i) {
       ItemType temp = A[i];
       A[i] = A[index_of_max];
       A[index_of_max ] = temp;
     }
  }
}
```

Khi sử dụng,có thể thay thế ItemType bằng int, string,... hoặc bất kỳ một đối tượng của một lớp nào đó

30

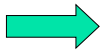
Ví dụ: Java dùng upcasting và Object

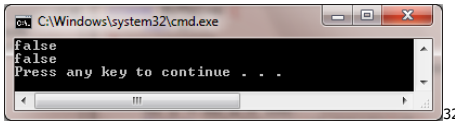
```
class MyStack {
  ...
  public void push(Object obj) {...}
  public Object pop() {...}
}
public class TestStack{
  MyStack s = new MyStack();
  Point p = new Point();
  Circle c = new Circle();
  s.push(p); s.push(c);
  Circle c1 = (Circle) s.pop();
  Point p1 = (Point) s.pop();
}
```

31

Nhắc lại – equals của lớp tự viết

```
class MyValue {
  int i;
}
public class EqualsMethod2 {
  public static void main(String[] args) {
     MyValue v1 = new MyValue();
     MyValue v2 = new MyValue();
     v1.i = v2.i = 100;
     System.out.println(v1.equals(v2));
     System.out.println(v1==v2);
  }
}
```





32

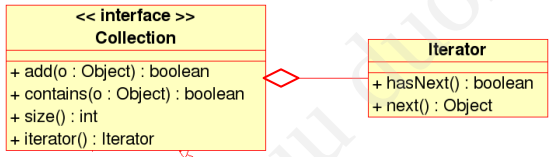
Bài tập

- Viết lại phương thức equals cho lớp MyValue (phương thức này kế thừa từ lớp Object)

```
class MyValue {
  int i;
  public boolean equals(Object obj) {
      return (this.i == ((MyValue) obj).i);
  }
}
public class EqualsMethod2 {
  public static void main(String[] args) {
    MyValue v1 = new MyValue();
    MyValue v2 = new MyValue();
    v1.i = v2.i = 100;
    System.out.println(v1.equals(v2));
    System.out.println(v1==v2);
  }
}
```

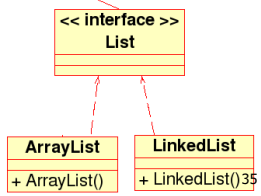
34

Ví dụ: Java 1.5: Template



- Không dùng Template

```
List myList = new LinkedList();
myList.add(new Integer(0));
Integer x = (Integer)
      myList.iterator().next();
```



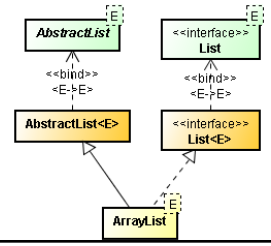
35

Ví dụ: Java 1.5: Template (2)

- Dùng Template:

```
List<Integer> myList = new LinkedList<Integer>();
myList.add(new Integer(0));
Integer x = myList.iterator().next();

//myList.add(new Long(0)); → Error
```



36

4. Lập trình tổng quát

- 4.1. Giới thiệu
- **4.2. Java generic data structure**
  - **4.2.1. Data structure**
  - 4.2.2. Java collection framework
  - 4.2.3. Các interface trong Java collection framework
  - 4.2.4. Các cài đặt cho các interface – implementation
- 4.3. Định nghĩa và sử dụng Template
- 4.4. Ký tự đại diện (Wildcard)

37

4.2.1. Cấu trúc dữ liệu-data structure

- Cấu trúc dữ liệu là cách tổ chức dữ liệu để giải quyết vấn đề.
- Một số cấu trúc dữ liệu phổ biến:
  - Mảng (Array)
  - Danh sách liên kết (Linked List)
  - Ngăn xếp (Stack)
  - Hàng đợi (Queue)
  - Cây (Tree)

38

a. Linked List

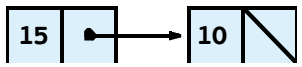
- Linked list là cấu trúc gồm các node liên kết với nhau thông qua các mối liên kết. Node cuối linked list được đặt là null để đánh dấu kết thúc danh sách.
- Linked list giúp tiết kiệm bộ nhớ so với mảng trong các bài toán xử lý danh sách.
- Khi chèn/xoá một node trên linked list, không phải dãn/dồn các phần tử như trên mảng.
- Việc truy nhập trên linked list luôn phải tuần tự.

39

a. Linked List (2)

- Thể hiện Node thông qua lớp tự tham chiếu (self-referential class)

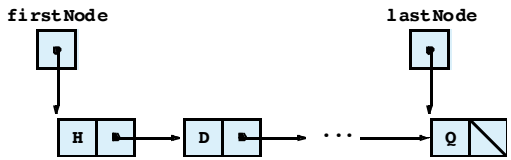
```
class Node
{
    private int data;
    private Node nextNode;
    // constructors and methods ...
}
```



40

a. Linked List (3)

- Một linked list được quản lý bởi tham chiếu tới node đầu và node cuối.



41

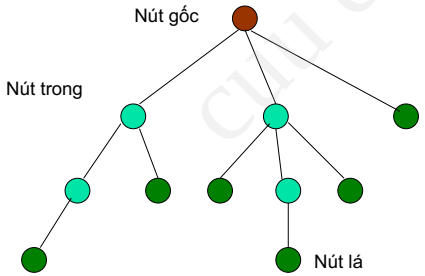
b. Stack

- Stack là một cấu trúc theo kiểu LIFO (Last In First Out), phần tử vào sau cùng sẽ được lấy ra trước.
- Hai thao tác cơ bản trên Stack
  - Chèn phần tử: Luôn chèn vào đỉnh Stack (push)
  - Lấy ra phần tử: Luôn lấy ra từ đỉnh Stack (pop)

42

c. Tree

- Tree là một cấu trúc phi tuyến (non-linear).
- Mỗi node trên cây có thể có nhiều liên kết tới node khác.



43

d. Queue

- Queue (Hàng đợi) là cấu trúc theo kiểu FIFO (First In First Out), phần tử vào trước sẽ được lấy ra trước.
- Hai thao tác cơ bản trên hàng đợi
  - Chèn phần tử: Luôn chèn vào cuối hàng đợi (enqueue)
  - Lấy ra phần tử: Lấy ra từ đầu hàng đợi (dequeue)

44

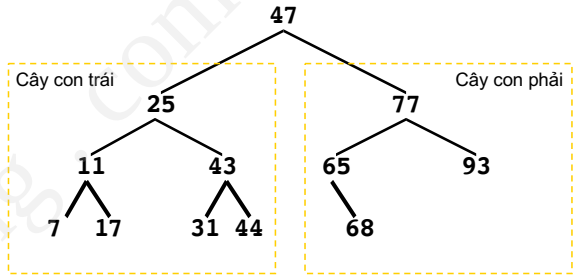
e. Binary Search Tree

- Cây nhị phân là cây mà mỗi node không có quá 2 node con.
- Cây tìm kiếm nhị phân là cây nhị phân mà:
  - Giá trị các nút thuộc cây con bên trái nhỏ hơn giá trị của nút cha.
  - Giá trị các nút thuộc cây con bên phải lớn hơn giá trị của nút cha.
- Duyệt cây nhị phân
  - Inorder traversal
  - Preorder traversal
  - Postorder traversal

45

e. Binary Search Tree (2)

- Ví dụ về Binary Search Tree



46

4. Lập trình tổng quát

- 4.1. Giới thiệu
- **4.2. Java generic data structure**
  - 4.2.1. Data structure
  - **4.2.2. Java collection framework**
  - 4.2.3. Các interface trong Java collection framework
  - 4.2.4. Các cài đặt cho các interface – implementation
- 4.3. Định nghĩa và sử dụng Template
- 4.4. Ký tự đại diện (Wildcard)

47

4.2.2. Java Collection Framework

- Collection là đối tượng có khả năng chứa các đối tượng khác.
- Các thao tác thông thường trên collection
  - Thêm/Xoá đối tượng vào/khỏi collection
  - Kiểm tra một đối tượng có ở trong collection không
  - Lấy một đối tượng từ collection
  - Duyệt các đối tượng trong collection
  - Xoá toàn bộ collection

48

4.2.2. Java Collection Framework (2)

- Các collection đầu tiên của Java:
  - Mảng
  - Vector: Mảng động
  - Hastable: Bảng băm
- Collections Framework (từ Java 1.2)
  - Là một kiến trúc hợp nhất để biểu diễn và thao tác trên các collection.
  - Giúp cho việc xử lý các collection độc lập với biểu diễn chi tiết bên trong của chúng.

49

4.2.2. Java Collection Framework (3)

- Một số lợi ích của Collections Framework
  - Giảm thời gian lập trình
  - Tăng cường hiệu năng chương trình
  - Dễ mở rộng các collection mới
  - Khuyến khích việc sử dụng lại mã chương trình

50

4.2.2. Java Collection Framework (4)

- Collections Framework bao gồm
  - Interfaces: Là các giao tiếp thể hiện tính chất của các kiểu collection khác nhau như List, Set, Map.
  - Implementations: Là các lớp collection có sẵn được cài đặt các collection interfaces.
  - Algorithms: Là các phương thức tĩnh để xử lý trên collection, ví dụ: sắp xếp danh sách, tìm phần tử lớn nhất...

51

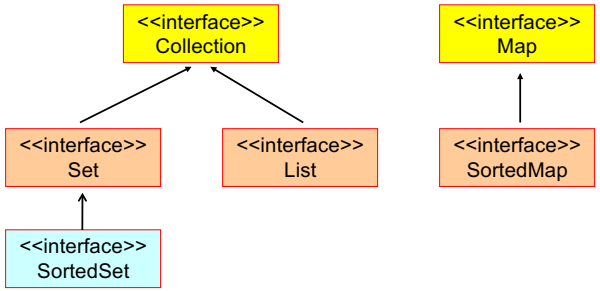
4. Lập trình tổng quát

- 4.1. Giới thiệu
- **4.2. Java generic data structure**
  - 4.2.1. Data structure
  - 4.2.2. Java collection framework
  - **4.2.3. Các interface trong Java collection framework**
  - 4.2.4. Các cài đặt cho các interface – implementation
- 4.3. Định nghĩa và sử dụng Template
- 4.4. Ký tự đại diện (Wildcard)

52

4.2.3. Interfaces

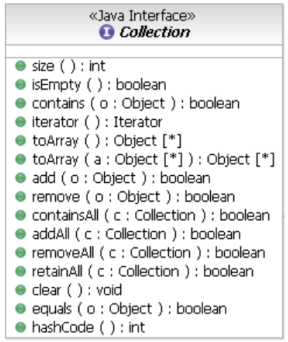
- List: Tập các đối tượng tuần tự, kế tiếp nhau, có thể lặp lại
- Set: Tập các đối tượng không lặp lại
- Map: Tập các cặp khóa-giá trị (key-value) và không cho phép khóa lặp lại



53

a. Giao diện Collection

- Xác định giao diện cơ bản cho các thao tác với một tập các đối tượng
  - Thêm vào tập hợp
  - Xóa khỏi tập hợp
  - Kiểm tra có là thành viên
- Chứa các phương thức thao tác trên các phần tử riêng lẻ hoặc theo khối
- Cung cấp các phương thức cho phép thực hiện duyệt qua các phần tử trên tập hợp (lặp) và chuyển tập hợp sang mảng



54

b. Giao diện List

- List kế thừa từ Collection, nó cung cấp thêm các phương thức để xử lý collection kiểu danh sách (Danh sách là một collection với các phần tử được xếp theo chỉ số).
- Một số phương thức của List
  - Object get(int index);
  - Object set(int index, Object o);
  - void add(int index, Object o);
  - Object remove(int index);
  - int indexOf(Object o);
  - int lastIndexOf(Object o);

55

c. Giao diện Set

- Set kế thừa từ Collection, hỗ trợ các thao tác xử lý trên collection kiểu tập hợp (Một tập hợp yêu cầu các phần tử phải không được trùng lặp).
- Set không có thêm phương thức riêng ngoài các phương thức kế thừa từ Collection.

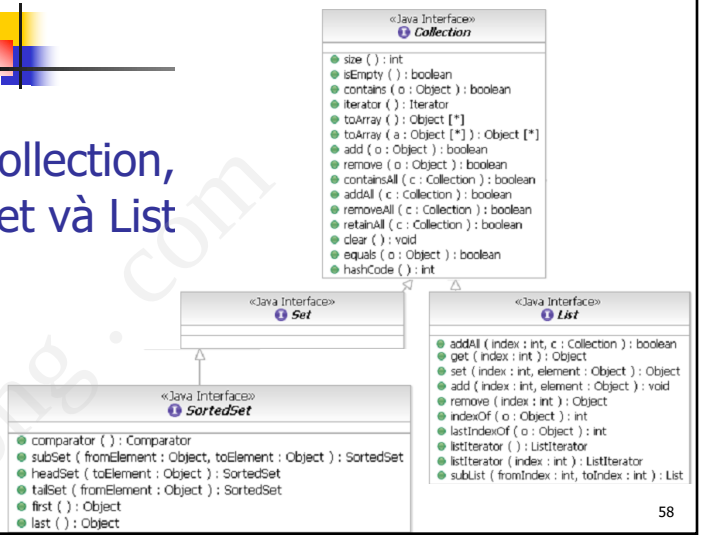
56

d. Giao diện SortedSet

- SortedSet kế thừa từ Set, nó hỗ trợ thao tác trên tập hợp các phần tử có thể so sánh được. Các đối tượng đưa vào trong một SortedSet phải cài đặt giao tiếp Comparable hoặc lớp cài đặt SortedSet phải nhận một Comparator trên kiểu của đối tượng đó.
- Một số phương thức của SortedSet:
  - Object first(); // lấy phần tử đầu tiên (nhỏ nhất)
  - Object last(); // lấy phần tử cuối cùng (lớn nhất)
  - SortedSet subSet(Object e1, Object e2); // lấy một tập các phần tử nằm trong khoảng từ e1 tới e2.

57

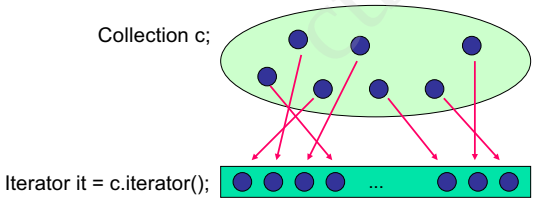
Collection, Set và List



58

e. Duyệt collection

- Các phần tử trong collection có thể được duyệt thông qua Iterator.
- Các lớp cài đặt Collection cung cấp phương thức trả về iterator trên các phần tử của chúng.



59

e. Duyệt collection (2)

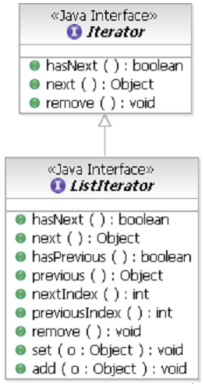
- Iterator cho phép duyệt tuần tự một collection.
- Các phương thức của Iterator:
  - boolean hasNext();
  - Object next();
  - void remove();
- Ví dụ:

```
Iterator it = c.iterator();
while ( it.hasNext() ) {
    Point p = (Point) it.next();
    System.out.println( p.toString() );
}
```

60

f. Giao diện Iterator

- Cung cấp cơ chế thuận tiện để duyệt (lặp) qua toàn bộ nội dung của tập hợp, mỗi lần là một đối tượng trong tập hợp
  - Giống như SQL cursor
- ListIterator thêm các phương thức đưa ra bản chất tuần tự của danh sách cơ sở
- Iterator của các tập hợp đã sắp xếp duyệt theo thứ tự tập hợp



f. Giao diện Iterator (2) - Ví dụ

```
Collection c;
// Some code to build the
  collection

Iterator i = c.iterator();
while (i.hasNext()) {
  Object o = i.next();
  // Process this object
}
```

62

g. Giao diện Map

- Xác định giao diện cơ bản để thao tác với một tập hợp bao gồm cặp khóa-giá trị
  - Thêm một cặp khóa-giá trị
  - Xóa một cặp khóa-giá trị
  - Lấy về giá trị với khóa đã có
  - Kiểm tra có phải là thành viên (khóa hoặc giá trị)
- Cung cấp 3 cách nhìn cho nội dung của tập hợp:
  - Tập các khóa
  - Tập các giá trị
  - Tập các ánh xạ khóa-giá trị

63

g. Giao tiếp Map (2)

- Map cung cấp 3 cách view dữ liệu:
  - View các khoá:
    Set keySet(); // Trả về các khoá
  - View các giá trị:
    Collection values(); // Trả về các giá trị
  - View các cặp khoá-giá trị
    Set entrySet(); // Trả về các cặp khoá-giá trị
- Sau khi nhận được kết quả là một collection, ta có thể dùng iterator để duyệt các phần tử của nó.

64

h. Giao diện SortedMap

- Giao diện SortedMap kế thừa từ Map, nó cung cấp thao tác trên các bảng ánh xạ với khoá có thể so sánh được.
- Giống như SortedSet, các đối tượng khoá đưa vào trong SortedMap phải cài đặt giao tiếp Comparable hoặc lớp cài đặt SortedMap phải nhận một Comparator trên đối tượng khoá.

65

4. Lập trình tổng quát

- 4.1. Giới thiệu
- **4.2. Java generic data structure**
  - 4.2.1. Data structure
  - 4.2.2. Java collection framework
  - 4.2.3. Các interface trong Java collection framework
  - **4.2.4. Các cài đặt cho các interface – implementation**
- 4.3. Định nghĩa và sử dụng Template
- 4.4. Ký tự đại diện (Wildcard)

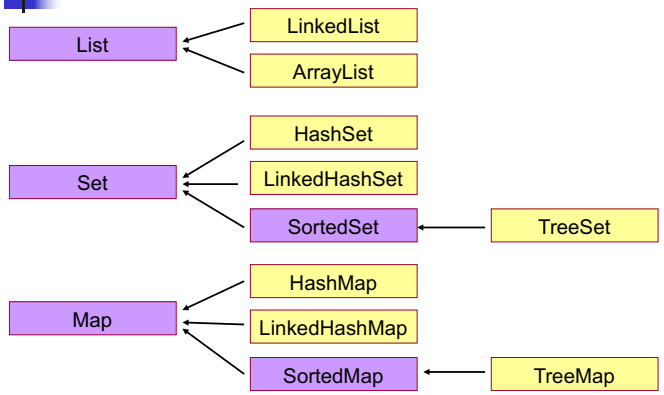
66

4.2.4. Implementations

- Các cài đặt trong Collections Framework chính là các lớp collection có sẵn trong Java. Chúng cài đặt các collection interface ở trên để thể hiện các cấu trúc dữ liệu cụ thể. Ví dụ: mảng động, danh sách liên kết, cây đỏ đen, bảng băm...

67

4.2.4. Implementations (2)



68

4.2.4. Implementations (3) -Mô tả các cài đặt

- ArrayList: Mảng động, nếu các phần tử thêm vào vượt quá kích cỡ mảng, mảng sẽ tự động tăng kích cỡ.
- LinkedList: Danh sách liên kết 2 chiều. Hỗ trợ thao tác trên đầu và cuối danh sách.
- HashSet: Bảng băm.
- LinkedHashSet: Bảng băm kết hợp với linked list nhằm đảm bảo thứ tự các phần tử.
- TreeSet: Cây đỏ đen (red-black tree).

69

4.2.4. Implementations (3) -Mô tả các cài đặt

- HashMap: Bảng băm (cài đặt của Map).
- LinkedHashMap: Bảng băm kết hợp với linked list nhằm đảm bảo thứ tự các phần tử (cài đặt của Map).
- TreeMap: Cây đỏ đen (cài đặt của Map).

70

4.2.4. Implementations (3) – Tổng kết

		IMPLEMENTATIONS				
		Hash Table	Resizable Array	Balanced Tree	Linked List	Legacy
INTERFACES	Set	HashSet		TreeSet		
	List		ArrayList		LinkedList	Vector, Stack
	Map	HashMap		TreeMap		HashTable, Properties

```
public class MapExample {
  public static void main(String args[]) {
      Map map = new HashMap();
      Integer ONE = new Integer(1);
      for (int i=0, n=args.length; i<n; i++) {
        String key = args[i];
        Integer frequency =(Integer)map.get(key);
        if (frequency == null) { frequency = ONE; }
        else {
          int value = frequency.intValue();
          frequency = new Integer(value + 1);
        }
        map.put(key, frequency);
      }
      System.out.println(map);
      Map sortedMap = new TreeMap(map);
      System.out.println(sortedMap);
  }
}
```

72

4. Lập trình tổng quát

- 4.1. Giới thiệu
- 4.2. Java generic data structure
  - 4.2.1. Data structure
  - 4.2.2. Java collection framework
  - 4.2.3. Các interface trong Java collection framework
  - 4.2.4. Các cài đặt cho các interface – implementation
- **4.3. Định nghĩa và sử dụng Template**
- 4.4. Ký tự đại diện (Wildcard)

73

4.3. Định nghĩa và sử dụng Template

```
class MyStack<T> {
    ...
    public void push(T x) {...}
    public T pop() {
        ...
    }
}
```

74

Nhắc lại ví dụ: Lập trình tổng quát trên Java dùng upcasting và Object

```
class MyStack {
  ...
  public void push(Object obj) {...}
  public Object pop() {...}
}
public class TestStack{
  MyStack s = new MyStack();
  Point p = new Point();
  Circle c = new Circle();
  s.push(p); s.push(c);
  Circle c1 = (Circle) s.pop();
  Point p1 = (Point) s.pop();
}
```

75

Sử dụng template

```
public class Test {
  public static void main(String args[]) {

      MyStack<Integer> s1 = new MyStack<Integer>();
      s1.push(new Integer(0));
      Integer x = s1.pop();

      //s1.push(new Long(0)); → Error

      MyStack<Long> s2 = new MyStack<Long>();
      s2.push(new Long(0));
      Long y = s2.pop();

  }
}
```

76

Định nghĩa Iterator

```
public interface List<E>{
    void add(E x);
    Iterator<E> iterator();
}

public interface Iterator<E>{
    E next();
    boolean hasNext();
}
class LinkedList<E> implements List<E> {
// implementation
}
```

77

4. Lập trình tổng quát

- 4.1. Giới thiệu
- 4.2. Java generic data structure
  - 4.2.1. Data structure
  - 4.2.2. Java collection framework
  - 4.2.3. Các interface trong Java collection framework
  - 4.2.4. Các cài đặt cho các interface – implementation
- 4.3. Định nghĩa và sử dụng Template
- **4.4. Ký tự đại diện (Wildcard)**

78

4.4. Ký tự đại diện (Wildcard)

```
public class Test {
  public static void main(String args[]) {
     List<String> lst0 = new LinkedList<String>();
     //List<Object> lst1 = lst0; → Error
     //printList(lst0); → Error
  }

  void static printList(List<Object> lst) {
     Iterator it = lst.iterator();
     while (it.hasNext())
          System.out.println(it.next());
  }
}
```

79

Ví dụ: Sử dụng Wildcards

```
public class Test {
  void printList(List<?> lst) {
     Iterator it = lst.iterator();
     while (it.hasNext())
        System.out.println(it.next());
  }

  public static void main(String args[]) {
     List<String> lst0 =
                new LinkedList<String>();
     List<Employee> lst1 =
                new LinkedList<Employee>();

     printList(lst0);   // String
     printList(lst1);   // Employee
  }
}
```

80

Các ký tự đại diện Java 1.5

- "? extends Type": Xác định một tập các kiểu con của Type. Đây là wildcard hữu ích nhất.
- "? super Type": Xác định một tập các kiểu cha của Type
- "?": Xác định tập tất cả các kiểu hoặc bất kỳ kiểu nào.

81

Ví dụ wildcard (1)

```
public void printCollection(Collection c) {
   Iterator i = c.iterator();
   for(int k = 0;k<c.size();k++) {
      System.out.println(i.next());
   }
}
```

→ Sử dụng wildcard:

```
void printCollection(Collection<?> c) {
   for(Object o:c) {
      System.out.println(o);
   }
}
```

82

Ví dụ wildcard (2)

```
public void draw(List<Shape> shape) {
   for(Shape s: shape) {
      s.draw(this);
   }
 }
```

→ Khác như thế nào với:

```
public void draw(List<? extends Shape> shape) {
   // rest of the code is the same
}
```

83

Template Java 1.5 vs. C++

- Template trong Java không sinh ra các lớp mới
- Kiểm tra sự thống nhất về kiểu khi biên dịch
  - Các đối tượng về bản chất vẫn là kiểu Object

84